Report Number: WUCS-93-54

1993

# The Pessimism behind Optimistic Simulation

George Varghese, Roger D. Chamberlain, and William E. Weihl

In this paper we make an analogy between the time that storage must be maintained in an optimistic simulation and the blocking time in a conservative simulation. By exploring this analogy, we design two new Global Virtual Time (GVT) protocols for Time Warp systems. The first simple protocol is based on the null message scheme proposed for clock advancement in some conservative approaches; this yields what we call Local Guaranteed Time. Our main contribution is a second new protocol that is inspired by Misra's circulating marker scheme for deadlock recovery in conservative simulations, and appears to have advantages over... **Read complete abstract on page 2.**

# The Pessimism behind Optimistic Simulation

George Varghese, Roger D. Chamberlain, and William E. Weihl

Complete Abstract:

In this paper we make an analogy between the time that storage must be maintained in an optimistic simulation and the blocking time in a conservative simulation. By exploring this analogy, we design two new Global Virtual Time (GVT) protocols for Time Warp systems. The first simple protocol is based on the null message scheme proposed for clock advancement in some conservative approaches; this yields what we call Local Guaranteed Time. Our main contribution is a second new protocol that is inspired by Misra's circulating marker scheme for deadlock recovery in conservative simulations, and appears to have advantages over previous GVT algorithms. It is simple enough to be implemented in hardware, takes no overhead in the normal path, can be made to work over non-FIFO links, and its overhead can be dynamically tuned based on computational load.

# The Pessimism behind Optimistic Simulation

**George Varghese**
**Roger D. Chamberlain**
**William E. Weihl**

**WUCS-93-54**

December 1993

In this paper we make an analogy between the time that storage must be maintained in an optimistic simulation and the blocking time in a conservative simulation. By exploring this analogy, we design two new Global Virtual Time (GVT) protocols for Time Warp systems. The first simple protocol is based on the null message scheme proposed for clock advancement in some conservative approaches; this yields what we call Local Guaranteed Time. Our main contribution is a second new protocol that is inspired by Misra's circulating marker scheme for deadlock recovery in conservative simulations, and appears to have advantages over previous GVT algorithms. It is simple enough to be implemented in hardware, takes no overhead in the normal path, can be made to work over non-FIFO links, and its overhead can be dynamically tuned based on computational load.

Department of Computer Science
Washington University
Campus Box 1054
One Brookings Drive
St. Louis, MO 63130-4899

# The Pessimism behind Optimistic Simulation

George Varghese* Roger Chamberlain † William E. Weihl ‡

December 22, 1993

**Abstract**

In this paper we make an analogy between the time that storage must be maintained in a optimistic simulation and the blocking time in a conservative simulation. By exploring this analogy, we design two new Global Virtual Time (GVT) protocols for Time Warp systems. The first simple protocol is based on the null message scheme proposed for clock advancement in some conservative approaches; this yields what we call Local Guaranteed Time. Our main contribution is a second new protocol that is inspired by Misra's circulating marker scheme for deadlock recovery in conservative simulations, and appears to have advantages over previous GVT algorithms. It is simple enough to be implemented in hardware, takes no overhead in the normal path, can be made to work over non-FIFO links, and its overhead can be dynamically tuned based on computational load.

## 1 Introduction

In distributed discrete-event simulation a physical system consisting of a set of *physical processes* is mapped onto a set of *logical processes (LPs)*. The actions of each $LP$ are then executed in parallel on a multiprocessor system; the simulation proceeds by having each $LP$ simulate the execution of the corresponding physical processes and send time-stamped messages to other $LP$s (that correspond to messages sent by the physical process). Each $LP$ $i$ also maintains a local clock $C_i$ that reflects the simulation time of that process.

*Dept. of Computer Science, Washington University, St. Louis. Work done while at Lab. for Computer Science, MIT. Supported by a graduate fellowship from DEC.

†Dept. of Electrical Engineering, Washington University, St. Louis. Supported in part by the National Science Foundation under Grant No. MIP-9309658.

‡Lab. for Computer Science, MIT. Supported in part by the Advanced Research Projects Agency under Contract N00014-91-J-1698, by grants from IBM and AT&T, and by an equipment grant from DEC.

In this paper we focus on the similarities between two popular clock advancement schemes: *conservative* [3, 6] and *optimistic* [7]. In particular, we explore the connection between *clock advancement* schemes in conservative approaches and *storage reclaiming* in optimistic schemes. We then exploit this connection to devise a new Global Virtual Time (GVT) algorithm for the archetypal optimistic simulation, the *Time Warp* system.

In the conservative approach, each $LP$ processes messages in strictly non-decreasing order, preserving causality constraints at all times. Thus messages sent by an $LP$ on any link have strictly non-decreasing time-stamps. This *safety* condition is enforced by advancing the local clock $C_i$ to the smallest (across all neighbors $j$ of $i$) of the latest time stamp received from neighbor $j$. Unfortunately, this simple local rule (called the input waiting rule) can lead to blocking and even deadlock; additional techniques are needed to prevent (or detect and resolve) deadlock [12].

In the optimistic approach by contrast, simulation messages are processed immediately; but if a later simulation message $m$ is received by $LP\ i$ with a time stamp less than $C_i$, then $LP\ i$ executes a *rollback*. The rollback restores the state of $LP\ i$ to an earlier state so that message $m$ can be processed. Thus each $LP\ i$ must save state so that it can rollback. If the simulation runs for a long time, and memory is finite, then saved state must be reclaimed. Thus it is necessary to periodically compute a bound (called GVT) such that any state that has time stamp less than GVT can be discarded.

Besides the core idea, both simulation approaches need auxiliary mechanisms for performance and indeed for correctness: the primary auxiliary mechanism in conservative schemes is for deadlock avoidance/detection and the primary auxiliary mechanism in optimistic schemes is for GVT calculation and memory reclamation. The principal thesis of this paper is that there is a simple analogy between the two auxiliary schemes for two widely contrasting approaches to simulation; moreover, this analogy can be exploited with profit. We demonstrate this by exhibiting two GVT algorithms (for *optimistic* schemes) that are derived from deadlock resolution algorithms (for *conservative* schemes). Our first GVT protocol illustrates the idea very simply but is not very efficient; our second GVT protocol appears to be competitive with the best GVT protocols that are known.

For the sequel we will use mnemonic archetypes. Thus, we use TW (for Time Warp) to denote an optimistic scheme and CM (for Chandy-Misra) to denote a conservative scheme.

## 2 Connection Between CM and TW Schemes

CM is conservative about advancing the local clock $C_i$ of $LP$ $i$. TW is conservative about reclaiming storage at $LP$ $i$. Put another way, CM waits until it can prove that $C_i$ will not fall below $t$ before setting $C_i$ to $t$. TW waits until it can prove that $C_i$ will not fall below $t$ before reclaiming storage with time stamp earlier than $t$. One can make an analogy between clock advancement schemes in CM (e.g., null messages, deadlock recovery) and GVT calculation schemes in TW. Similarly one can make an analogy between the time spent waiting for the local clock to advance in CM, and the time spent waiting for storage to be reclaimed in TW.

Philosophically this analogy favors TW. First, memory is cheap and easy to build; current VLSI trends make it attractive to trade large amounts of memory for a small decrease in processing power. Second, the overhead of state reclaiming in TW can be amortized over several simulation events; on the other hand, in some CM simulations, the global overhead of clock advancement must be paid for almost every simulation event. We note, however, that this is only a philosophical comparison of one aspect of the two approaches: there are many more factors that must be considered in comparing the two schemes.

But more concretely, this analogy suggests the following idea. *Any clock advancement scheme for CM should be considered, with suitable modifications, as a possible candidate for GVT calculation in TW.* In the sequel, we explore this simple idea.

There is some theoretical basis for this analogy. Gerhard Tel [14] has shown that both termination detection and GVT calculation are special cases of what he calls *distributed infimum approximation.* Indeed, it seems apparent from Tel's result that every GVT protocol can be converted (in a precise sense) into a deadlock detection protocol. Thus, for instance, Samadi's GVT protocol [13] can be modified for use as a deadlock detection protocol.

However, it by no means follows from Tel's result that every deadlock detection protocol can be converted into a GVT protocol. Thus the results in this paper *are not trivialized by Tel's result.* Perhaps (in Tel's terminology) our results can be described as a search for the distributed infimum algorithms underlying certain deadlock detection schemes.

The rest of the paper is organized as follows. In Section 3 we present a simple model of a TW system and in Section 4 we describe previous approaches to calculating GVT. Then in Section 5 we present a simple, quick example of how the null message scheme used in CM can be modified to reclaim storage in TW. However, in many cases the null message scheme is too slow. Thus in Section 6 we consider the deadlock detection scheme of [11] as a point of departure. Then in Section 7 we present our new (and we believe competitive) GVT algorithm. Rather than present the complete GVT scheme, we present a series of refinements. We start

with a scheme that works only for a simplified model of TW and over FIFO links; we then refine the scheme to work with non-FIFO links and a more accurate model. Finally, we describe modifications to make the protocol faster and load-adjusting.

# 3 Model of a TW System

Our ideas appear to apply to *any* optimistic distributed simulation scheme. However, we will use a model that essentially reflects a Time Warp setting.

Thus in a TW system, every message $m$ carries a time stamp $ts(m)$ and every $LP$ $i$ has a local clock called $C_i$. For our purposes, TW works as follows. Processing at an $LP$ (after initialization) is message-driven; locally generated events are simulated by an $LP$ sending a message to itself. On receiving a message $m$ from the Data Link (typically as part of an interrupt routine), $LP$ $i$ behaves as follows. If $ts(m) \geq C_i$, then $m$ is queued in an input queue for processing according to some queueing discipline. If $ts(m) < C_i$ then a rollback occurs: $LP$ $i$ aborts its current processing, does some rollback recovery, sets $C_i = ts(m)$, and then begins to process $m$. Except for interruptions to accept new messages and do background processing, $LP$ $i$ stays in a loop processing messages from the input queue.

As part of processing a message $m_1$, $LP$ $i$ may send other messages, including say $m_2$, to a neighbor $LP$ $j$. $LP$ $i$ sends $m_2$ by giving it to the Data Link for link $(i, j)$. TW *requires* that $ts(m_2) \geq ts(m_1)$. Data links are reliable, but are not necessarily FIFO. A message $m$ is *in transit* if $m$ has been given to the Data link for $(i, j)$ but has not been delivered to $LP$ $j$,

Global Virtual Time (henceforth GVT) at any instant of a TW simulation is defined to be the minimum of $ts(m)$ and $C_i$, across all messages $m$ in transit and all $LP$s $i$. ¿From the previous discussion, it is easy to see that GVT is non-decreasing, and represents a value of simulation time below which the simulation will never rollback. When its input queue is empty, $LP$ $i$ signals local termination by setting $C_i = \infty$; the simulation terminates when GVT $= \infty$.

Some Time Warp systems use a slightly more sophisticated definition of GVT that incorporates flow control of messages. We prefer to start with the simple definition and describe the modifications required for the more sophisticated definition in Section 7.6.

GVT calculation is important for three reasons. First it allows state to be reclaimed. When $LP$ $i$ finds that GVT is $G$, it can find the latest state $S$ that it has saved with time stamp less than $G$. Suppose the time stamp associated with $S$ is $t$. Then $LP$ $i$ can garbage collect all saved state and messages with time stamps less than $t$. Second, GVT calculation allows outputs to be committed. Only output messages (e.g., to a terminal) with time stamps less than GVT can be safely released. Thirdly, GVT is used to detect termination.

Thus the faster GVT is updated, the faster state can be reclaimed, outputs can be released,[1] and termination detected. On the other hand, GVT calculation costs overhead. This causes a significant problem. To quote from Jefferson [8]:

> ". . . we set the interval between GVT calculation at 5 seconds, except that toward the end of a run it was reduced to 1 second. GVT calculation is a significant source of overhead, . . . but since termination can only be detected when GVT is updated, if we retained the 5 second interval to the end of a run our uncertainty in the time of termination would be as much as 5 seconds . . . "

Thus there is considerable incentive to find a fast, low-overhead GVT algorithm.

# 4 Existing Approaches

If all network links are FIFO and bi-directional, then the celebrated Chandy-Lamport snapshot algorithm [5] can be used to obtain a lower bound on GVT. However, the designers of Time Warp believe that non-FIFO links improve the performance of TW: they argue that TW processes messages out of order anyway. The Chandy-Lamport snapshot algorithm has other disadvantages which we discuss later.

A commonly used algorithm for GVT appears to be Samadi's algorithm [13], which works as follows. First, every message sent by the simulation is acked after being received by the receiving $LP$. This ack is *in addition to any acknowledgements used by the Data Link for reliable delivery*; the ack must be at the simulation layer for Samadi's algorithm to be correct. These acks allow an $LP$ to keep track of the messages in transit on a link. Periodically a specified coordinator $LP$ initiates a GVT calculation by broadcasting a query to every $LP$. On getting a query, $LP\ i$ reports a value that is the smallest of three values. For our purposes, the most significant of these three values is the minimum time stamp of all unacknowledged messages sent by $LP\ i$. Computing this quantity is the bottleneck of query processing.

In Samadi's scheme, $LP\ i$ searched the entire queue of saved messages sent since the last GVT calculation, an enormous overhead. Bellenot [10] described a simple optimization: he suggested keeping track of the unacknowledged messages separately using a doubly-linked list. A message is removed from this list when it is acked. Nevertheless, there are several disadvantages to Samadi's scheme, even with Bellenot's optimization:

[1] This is especially important for interactive simulations.

- It adds significant overhead to the processing of simulation messages in order to send and process acks. The extra simulation-level acks could also slow down the network links.[2] Note that Bellenot's suggestion adds even more overhead to the normal path than Samadi's original algorithm.

- The actual query processing takes significant overhead that is proportional (even with Bellenot's optimization) to the number of unacknowledged messages.

- It requires the use of ad hoc timers at the coordinator to control GVT calculation overhead. As described earlier, these timers need to be adjusted in an ad hoc way to detect termination faster.

# 5 TW Using Local Guaranteed Time

For our first GVT algorithm, we will exploit the theory associated with null message deadlock avoidance in CM simulation schemes. Essentially, in conservative algorithms the input waiting rule allows us to determine a safe $C_i$ such that no later arriving message $m$ will have a time stamp $t$ less than $C_i$.

For TW, we introduce the concept of a *local guaranteed time*, or LGT, that follows the input waiting rule of CM. LGT is defined for $LP\ i$ as the minimum of two quantities: the LGT (plus lookahead capability) of each $LP$ that might send $LP\ i$ a message; and the time stamp of each message $m$ currently in transit to $LP\ i$.

The significance of LGT is identical to GVT, no rollback can occur to a time less than LGT, and therefore all saved state and messages with time stamps before LGT can be reclaimed. Unlike GVT, LGT is local to an $LP$. Different $LP$s will, in general, have different values for LGT. GVT is now the minimum LGT over all the $LP$s.

The advantage of using LGT instead of GVT arises from the fact that different processors can have different values for LGT, and hence some of them can have values for LGT that are bigger than GVT. $LP$s with LGT strictly greater than GVT will have lower memory requirements, and $LP$s with LGT equal to GVT will have the same memory requirements. The total memory requirements will therefore decrease.

For the calculation of LGT, we propose an algorithm that is a variant on the conservative time advance mechanism using null messages [12]. Just as null messages exploit the minimum

[2] Lin[10] suggests a variant of Samadi's scheme that gets rid of the acks but at the cost of even more processing. Lin's scheme also needs statistical assumptions in order to have good performance.

lookahead that is known for a given application to advance the clock in a conservative simulation, we can use exactly the same idea in an optimistic simulation to calculate LGT. The code for this implementation is in the appendix.

Unfortunately, null message schemes work poorly if the ratio of the lookahead to the average time between simulation events is small, and if the time to send a message between $LP$s is large. There are a number of real examples (see [4]) where null messages do badly; examples include circuit simulations or simulations of computer networks that contain cycles. Thus while this scheme furnishes a simple example of the analogy between clock advancement and GVT calculation, we will consider a better scheme in the next few sections.

# 6 Misra's Deadlock Detection Scheme

As a point of departure for our second GVT scheme, we use the following scheme for deadlock detection in CM. Assume all links are FIFO and unidirectional; a full duplex link consists of two unidirectional links in opposite directions. Deadlock occurs when every $LP$ in the CM simulation is idle and there are no messages in transit. In Misra's scheme [11], a marker circulates around the network in a tour. The tour is such that it traverses each network link at least once.[3] Whenever an $LP$ receives a message it updates a status bit to "busy;" whenever a marker leaves an $LP$, the $LP$ resets its status bit to "idle." The marker detects deadlock if it finds that at the end of a tour every $LP$ it visited in the tour was idle.

Below, we modify Misra's circulating marker scheme to calculate GVT. We then extend the basic scheme (and Misra's as well) by making it faster, and allowing it to handle non-FIFO links.

# 7 New GVT Algorithm

As in Misra's scheme there is a marker that continuously circulates in a tour such that in one tour it traverses each unidirectional network link. The tour starts from some originator $s$ and returns to $s$ at the end of the tour. After a marker visits an $LP$, the marker may remain in that $LP$ for an arbitrary but finite time before it is processed and sent to the next $LP$ in the tour. Let $T_r$ denote the $r$-th tour.[4]

For now assume that links are FIFO. The basic scheme is modified below to handle non-FIFO links. Each $LP$ $i$ keeps a variable $Local\_Min_i$ that is initially 0; $Local\_Min_i$ stores the

[3] For some graphs, it is possible to use an Euler tour that traverses each link exactly once.

[4] $r$ denotes a round number; each tour completes a round.

lowest value of $C_i$ that has occured since the last time the marker visited $LP\ i$. Thus $Local\_Min_i$ needs to be updated only when a marker is processed at an $LP$ (upon which $Local\_Min_i$ is reset to $C_i$) and when a rollback occurs at $LP\ i$ (upon which $Local\_Min_i$ is is set to the smaller of its old value and the new value of $C_i$).

The marker has a field called *Current_Min* which stores the smallest value of $Local\_Min_i$ for every $LP$ that the marker has encountered so far in the current tour. Note that an $LP$ may be visited several times, and the minimum is taken across all visits to the $LP$. When the marker returns to $s$, $s$ uses the value of *Current_Min* (after possibly updating *Current_Min* to incorporate the value of $Local\_Min_s$) in the marker as the new GVT estimate. Initially all $LP$s set GVT to 0.

To broadcast the value of GVT to every $LP$ a marker also carries a GVT field that carries the value calculated by the originator $s$ during the last tour. When an $LP\ i$ (other than the originator) receives a marker, $LP\ i$ stores the marker. Some time later when it processes the marker:

- $LP\ i$ updates its estimate of GVT using the GVT field in the marker.
- $LP\ i$ updates the *Current_Min* field in the marker to be the smaller of *Current_Min* and $Local\_Min_i$.
- $LP\ i$ sets $Local\_Min_i$ to $C_i$ and sends the marker to the next $LP$ in the tour.

Thus marker processing consists of three simple operations. The originator $s$ is slightly different. When $s$ sends a marker out to initiate a tour, it sets the *Current_Min* field to $\infty$, the GVT field to its current estimate of GVT, and sets $Local\_Min_s$ to $C_s$. On receiving and processing a marker for the last time in the tour, $s$ updates its value of GVT to be the smaller of the *Current_Min* field in the marker and $Local\_Min_s$. Thus the processing at the originator is also very simple.

### 7.1 GVT algorithm correctness

The following theorem expresses the correctness of the algorithm. Let $Est(T_r)$ be the estimate of GVT calculated at the end of the $r$-th tour. The first part of theorem says that the estimate calculated is a lower bound on the real GVT. The second part of the theorem shows that the estimate is a non-trivial lower bound, because the estimate is at least as large as the GVT at the start of the previous round, assuming such a round exists.

**Theorem 7.1** *The following bounds hold for $Est(T_r)$:*

- *$Est(T_r)$ is no greater than the GVT at the start of $T_r$.*
- *$Est(T_r)$ is no less than the GVT at the start of $T_{r-1}$, for $r > 1$.*

**Proof:** we prove each part separately.

- Let the real GVT at the start of $T_r$ be $t$. There are two cases. Suppose there is some $LP\ i$ that has $C_i = t$ at the start of $T_r$. Then it is easy to see that $Local\_Min_i \leq t$. Thus by the end of $T_r$, $Current\_Min \leq t$.

  On the other hand, if there is no $LP\ i$ at the start of $T_r$ with $C_i = t$, then (by definition of GVT) there must be some message $m$ with $ts(m) = t$ in transit to say $LP\ j$. Then before the end of $T_r$, because links are FIFO, message $m$ will be received by $LP\ j$. Let us denote by $\tau$ the real time at which message $m$ is received by $LP\ j$. Clearly $Local\_Min_j \leq t$ in the interval starting from real time $\tau$ until the next time a marker is processed at $j$. And we know that the marker must visit $j$ at least once more (i.e., over the link on which $m$ was received) after $\tau$ and before the end of $T_r$. Let $\tau'$ be the first time after $\tau$ that the marker is processed at $j$. Then clearly from real time $\tau'$ until the end of tour $T_r$, the *Current_Min* field in the marker will be no greater than $t$.

- If $Est(T_r) = t$, then there must have been some real time $\tau$ (that lies between the start and end of tour $T_r$) and some $LP\ i$ such that: a) the marker is processed at $LP\ i$ at real time $\tau$, and b) the value of $Local\_Min_i$ at real time $\tau$ is $t$. Intuitively, this is because the marker estimate collects the lowest of the $Local\_Min_i$ values it encounters during tour $T_r$.

  Assume that $r > 1$ (i.e., there has been a tour before $T_r$). Let $\tau'$ be the last (real) time before $\tau$ that $LP\ i$ received a marker. Now we know that $Local\_Min_i$ stores the lowest clock value at $LP\ i$ since the marker last visited. Thus there must be some real time $\tau''$ in the interval $[\tau', \tau]$ such that $C_i = t$ at real time $\tau''$. Let $GVT(\tau)$ denote the GVT at any real time $\tau$. Then by definition, $GVT(\tau'') \leq t$. Also since the marker must visit $LP\ i$ after the start of $T_{r-1}$, we know that $Start(T_{r-1}) \leq \tau''$, where $Start(T_{r-1})$ denotes the real time at which $T_{r-1}$ starts. Finally since GVT is non-decreasing we get:

  $GVT(Start(T_{r-1})) \leq GVT(\tau'') \leq t = Est(T_r)$, as we desire.

■

**Note:** A simple corollary to this theorem is that termination can be detected in at most two tours after termination actually occurs.

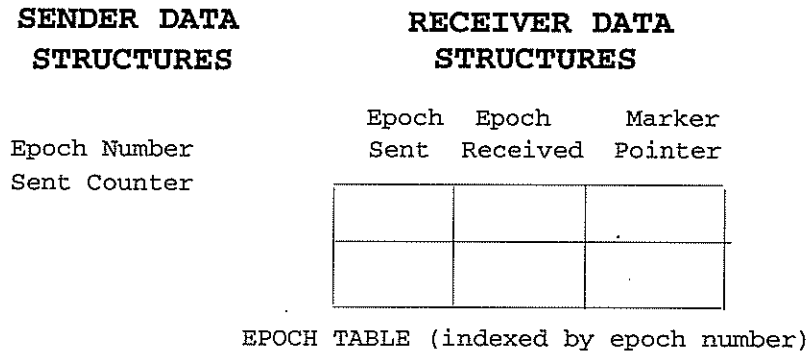


Figure 1: Key Data Structures for our Efficient, Problem Specific Flushing Protocol

### 7.2 Extension to non-FIFO links

It is crucial to extend any competitive GVT algorithm to work over non-FIFO links. Many real multiprocessors have interconnection networks that are reliable but non-FIFO. We now show how to extend our marker algorithm to non-FIFO links.

Let $(i, j)$ denote a link from *LP* $i$ to *LP* $j$. From the proof, all that is needed for our GVT algorithm to be correct is the following. *Before a marker is processed at LP* $j$*, all messages that were sent before the marker on link* $(i, j)$ *should be received at* $j$. Intuitively, the marker should "flush" the link before it is processed. This happens without extra effort on FIFO links.

A flush primitive [1] is a useful primitive for non-FIFO data links. The particular flush primitive we require is what is called [1] a "forward flush" primitive. We now show that this primitive can be efficiently implemented.

### 7.3 Implementing a flush primitive

We assume a non-FIFO but reliable network. Our solution is simpler than the ones suggested by Ahuja [1, 2, 9] because it exploits the semantics of the problem domain.

Consider any link $(i, j)$ where $i$ is the sender and $j$ the receiver. The basic idea is that each marker sent by $i$ is assigned a unique *EpochNumber*; the duration between the sending of two consecutive markers on link $(i, j)$ is referred to as an *epoch* on that link. Figure 1 shows the major data structures used by our implementation.

The sender $i$ keeps track of the number of simulation messages sent in each epoch (*SentCounter*) and the current *EpochNumber*. On sending a message $m$, *SentCounter* is incremented and $m$ is tagged with *EpochNumber*. Each time the sender sends a marker, the sender tags the marker

with the current *EpochNumber* and the current *SentCounter*. The sender then reinitializes the epoch variables by incrementing *EpochNumber* and setting *SentCounter* to zero.

The receiver keeps a table containing information about $w$ epochs, where $w$ is the maximum number of incomplete epochs. For a tour, observe that *a marker is sent again on a link only after all previous markers have been delivered.* Thus the receiver needs to keep track of at most two outstanding epochs: the epochs immediately before and after this marker. Hence we need at most two table entries. This is an enormous simplification because multiple outstanding markers on a link leads to increased storage and increased processing. Notice that we cannot deliver a marker till the current epoch *and* all previous epochs have completed.

Thus we assume that the *EpochNumber* is just a bit that gets toggled. The receiver keeps a table ( Figure 1) with two entries: the first for epochs with bit 0, the second for epochs with bit 1. Each entry contains 3 records. The first record is a variable *EpochSent*; if a marker for this epoch has been received, this variable is set to the *SentCounter* carried in this epoch. Similarly, the *MarkerPointer* points to a marker message $m$ if $m$ is the marker for this epoch, and $m$ has arrived but all the simulation messages in this epoch have not arrived; otherwise this value is undefined. Finally the *EpochReceived* variable counts the number of simulation messages that have arrived in this epoch. Initially (and after the completion of an epoch) *EpochReceived* is set to 0 and *EpochSent* is set to $-1$.

The receiver processing is simple. When a simulation message arrives carrying *EpochNumber* $e$, we deliver this message and increment *EpochReceived*$[e]$; next if *EpochReceived*$[e] =$ *EpochSent*$[e]$ we deliver the marker being pointed to by *MarkerPointer*$[e]$ and set *EpochSent*$[e] = -1$. Thus we only have a constant (small) number of operations.

When a marker $m$ arrives with *EpochNumber* $e$ and *SentCounter* $s$ we set *EpochSent*$[e] = s$. If *EpochReceived*$[e] =$ *EpochSent*$[e]$ we deliver the marker and reinitialize the epoch variables; otherwise we set *MarkerPointer*$[e]$ to point to $m$. Once again we only have a constant (small) number of operations.

Ahuja has a number of implementations of flush primitives [1, 2, 9] but our implementation is simpler.[5] This is because we exploit two features of our problem domain: we only need forward flushing, and we can have at most one marker at a time on any link.

## 7.4 Making it faster

One disadvantage of the marker scheme over Samadi's is that tours must traverse every link in the network. Consider a network with $M$ links and diameter $D$, and assume that markers are

[5] We are indebted to Mohan Ahuja for his clarifications on these points.

processed immediately. Then Samadi's algorithm will take $O(D)$ time to complete while the marker algorithm will take $O(M)$ time. Typically $D$ is much smaller than $M$. One can regard the extra time of our algorithm as a substitute for the 5 second timer run at the originator in Samadi's algorithm. But there is a simple way to make it faster.

The idea is that the source initiates a tour by sending several markers in parallel. All we need is that at least one marker flushes each link. The source now has to take the minimum of the *Current_Min* fields of all the markers it receives. Other $LP$s (besides the originator) can also create and collect markers at the cost of slightly increased work at these $LP$s. By using parallel markers, the algorithm can work in $O(D)$ time.

**Note:** Both the modification for non-FIFO links and the parallel marker scheme apply to Misra's original deadlock detection scheme.

### 7.5 Propagating markers: making it load-adjusting

Since the marker algorithm is computationally simple, we propose that the marker be run continuously without timers. However, the rate at which a marker is processed will vary with the load at an $LP$. The intent is that when $LP$s are idle (especially after termination) the marker moves very fast; otherwise the marker is held long enough to guarantee that $y$ units of "useful" computation is done at the $LP$ before the marker is processed. If $x$ is the amount of computation required to process the marker then $\frac{x}{x+y}$ is an upper bound on the overhead of the marker scheme. By appropriate choice of $y$, we hope it is possible to keep overhead small and yet provide quick turnaround. Another heuristic for propagating markers is to speed up marker processing when memory is running low, or when *Current_Min* is much higher than the last GVT estimate.

### 7.6 Incorporating flow control in GVT definition

Actual TW implementations [7] use a slightly different definition for Global Virtual Time that we will denote by $GVT'$. What we called the time stamp of a message $ts(m)$ is the simulation time that the message is to be processed at the receiver; this is also called the virtual receive time of $m$. Define $ts'(m)$ as the simulation time at the sender at the instant the message $m$ was sent; this is also called the virtual send time of $m$. A message $m$ will carry both $ts(m)$ and $ts'(m)$. Jefferson's definition of GVT uses $ts'(m)$ instead of $ts(m)$. This is because the flow control algorithm can cause a message $m$ at a receiver to be sent back to the sender, thus causing the sender to roll back to $ts'(m)$.

There are two possible modifications to the marker algorithm in order to calculate $GVT'$. Note that $GVT' \leq GVT$ at any instant of time. The first approach is to update $Local\_Min_i$ whenever any message $m$ is received (as opposed to only when a rollback occurs) using $ts'(m)$. This adds a small amount of work to the normal path. The other alternative is to realize that $GVT'$ at the start of round $r$, for $r > 1$, is no less than the value of $GVT$ at the start of round $r - 1$. Given this claim (which is stated without proof), the previous algorithm can remain unchanged except for the following. Each $LP$ keeps *two* $GVT$ estimates: one calculated on the last tour, and one calculated on the next-to-last tour. An $LP$ then uses the older estimate of $GVT$ to reclaim storage. As a consequence it will take three tours to detect termination.

## 7.7 Properties of the marker algorithm for GVT calculation

The marker algorithm

- Seems simple enough to be run continuously without timers, possibly even in hardware. It is simple because marker processing is fast (a few lines of code) and because only one message type is used.
- Has little or no overhead in the normal path. In the algorithm for FIFO links there is no extra work to process a simulation message in the normal path. The algorithm also only executes one line of extra code when a rollback occurs. Finally, the algorithm does not need to send or process acks.
- Can be run in the background - i.e., an $LP$ does not have to run the algorithm at the instant it gets a marker but can postpone working on it till convenient.
- Can be made load adjusting without timers - i.e., when $LP$s are idle, it runs very fast; when $LP$s are busy, the marker travels at a reasonable rate while maintaining some acceptable overhead.
- Works for any strongly connected topology (e.g., an FDDI ring) with non-FIFO links.
- Has a simple proof that allows variations to be designed easily.

By contrast, we have already listed the disadvantages of the Samadi scheme. The Chandy-Lamport algorithm on the other hand: a) works only for FIFO bidirectional links, the extension to non-FIFO links requires two-way flushing [1] that is more expensive to implement; b) appears to require that an $LP$ process a marker immediately upon receipt (this makes it hard to make the snapshot scheme load-adjusting); and c) requires overhead in the normal path to keep track of all messages received on a link till a marker arrives on the link.

# 8 Conclusions

Time Warp [7] introduced a novel package of mechanisms that showed that optimistic schemes for distributed simulation were feasible. However, as the field matures, it is important that these mechanisms be re-evaluated. In this paper we have concentrated on GVT algorithms. We have described a new GVT algorithm that seems practical and appears to be better than previous GVT algorithms. We have also presented a simple implementation of forward flushing (to incorporate non-FIFO links) that may be of independent interest.

We conclude with the following questions as possible directions for future work. The first two are questions about the marker scheme itself and have no relation to simulation.

**Can marker schemes be used for distributed databases?** : Distributed databases that use time stamps sometimes have a similar need to find the earliest time stamped message in the system in order to reclaim storage. Can a marker scheme offer advantages over a snapshot mechanism for this application?

**What other stable properties can be detected by markers?** : Misra uses markers to detect deadlock; we use them to detect lowest virtual time. It is well known that Chandy-Lamport snapshots can be used to detect any stable property. An obvious question is to characterize the stable properties that a circulating marker can detect.

**Can other deadlock detection algorithms be converted into GVT algorithms?** : There are several other deadlock detection schemes and it seems likely that some of them can be coverted into GVT algorithms as well.

**Are anti-messages really necessary?** : TW uses anti-messages to undo the effect of messages sent before a rollback. But it is easy to construct examples where TW generates an exponential number of anti-messages. These messages cost in two ways: first they add extra storage (TW has to stop if it runs out of storage) and second they cause processing overhead to generate and process. In fact when an anti-message arrives, a (seemingly costly) search must be done to search for its positive counterpart.

Samadi [13] points out that with FIFO delivery it suffices to send a single special message after a rollback that is time stamped with the rollback time. No anti-messages are necessary. More work is required to figure out the details, but Samadi's suggestion seems promising.[6] In particular, it can be extended to non-FIFO links using the flush primitive proposed originally by [1] and in this paper.

[6] Samadi does not pursue this approach.

Finally, we feel that there may be other connections between schemes used in TW and CM. We hope there will be more work in identifying and exploiting these connections. Our paper is a step in this direction.

# Appendix A TW Implementation Using LGT

This appendix describes a TW algorithm using local guaranteed time (LGT) in place of GVT. To facilitate the different rules that drive messages and local clocks in the conservative and optimistic algorithms, messages in the LGT algorithm have two time stamps, a *current time* and a *guaranteed time*. The current time stamp follows the rules in the optimistic algorithm (i.e., not required to be in order, can be deleted, etc.). The guaranteed time stamp follows the rules in the conservative algorithm (i.e., non-decreasing time stamp order, etc.).

With this approach, the guaranteed time stamp can be used to calculate LGT at each $LP$. LGT at $LP\ i$ is the minimum guaranteed time on the last message received from each $LP$ that might send $LP\ i$ a message. Notice that this is simply the input waiting rule for conservative simulation. If $LP\ i$ needs to advance its LGT, and no message is forthcoming from $LP\ j$ (the $LP$ that is limiting LGT at $LP\ i$), $LP\ i$ requests a null message from $LP\ j$ to advance LGT. By using null message deadlock avoidance techniques from the conservative simulation literature (e.g., see [12]), LGT can easily be seen to be both safe and deadlock free.

A top level view of the TW algorithm using LGT is given in Figure 2. The LGT calculation is performed on the line marked with an "*" character. Outgoing messages have their guaranteed time set to the LGT of $LP\ i$ plus the lookahead capability of $LP\ i$.

```
parallel (on each processor)
    while (termination criteria not met)
        while (no incoming msgs)
            process msgs from local queue, any outgoing msgs have
                guaranteed time of LGT + local lookahead capability
        endwhile
        receive msg
        if (msg is normal message) then
*           LGT ← guaranteed time (∀ incoming msg links)
            if (msg current time ≥ C_i) then
                enqueue msg in local queue
            else     # message time stamp < C_i
                rollback to current time of msg
                send out anti-msgs for time > rollback time
                C_i ← rollback time
            endif
        else     # message is anti-message
            rollback to current time of msg
            send out anti-msgs for time > rollback time
            C_i ← rollback time
        endif
    endwhile
endparallel
```

Figure 2: TW Algorithm Using LGT

## References


[1] Ahuja, M., *Flush* Primitives for Asynchronous Distributed Systems, *Information Processing Letters, pages 5–12*, 1990.

[2] Ahuja, M., An Implementation of F-Channels, *IEEE Transactions on Parallel and Distributed Systems, 4(6): 658–667*, 1993.

[3] R.E. Bryant, Simulation on a Distributed System, *in COMPSAC*, 1979.

[4] Roger Chamberlain, Analysis of Parallel Algorithms for Mixed-Mode Simulation, *D.Sc. Thesis, Dept. of Computer Science, Washington University, St. Louis*, 1989.

[5] K.M. Chandy and L. Lamport, Distributed Snapshots: determining global states of distributed systems, *ACM TOCS 3(1): 63–75*, Feb. 1985.

[6] K.M. Chandy and J. Misra, Asynchronous distributed simulation via a sequence of parallel computations. *CACM 24(11): 198–206*, April 1981.

[7] D. Jefferson, Virtual Time, *ACM POPL*, 7(3): 404–425, July 85.

[8] D. Jefferson., et al., Distributed Simulation and the Time Warp Operating System, *Proc. 11th SOSP*, pp. 77–93, Nov 1987.

[9] Kearns, P. and Camp, T. and Ahuja, M., Implementation of *Flush* Channels based on a verification methodology, *Proceedings Twelfth Int'l Conf. on Distributed Computing Systems, pages 336–343*, 1992.

[10] Yi-Bing Lin, Understanding the Limits of Optimistic and Conservative Parallel Simulation, *Ph.D. Thesis, Comp. Science Dept., University of Washington, Seattle*, Aug. 1990.

[11] J. Misra, Detecting Termination of distributed computations using markers, *Proc. 2nd ACM PODC*, pages 290–293, 1983.

[12] J. Misra, Distributed Discrete-Event Simulation, *Computing Surveys*, 18(1): 39–65, March 1986.

[13] B. Samadi, Distributed Simulation, Algorithms and Performance Analysis, *Ph.D. Thesis, Comp. Science Dept., UCLA, Los Angeles*, 1985.

[14] G. Tel, Topics in Distributed Algorithms, *Cambridge University Press*, 1991.